ASP.NET Core provides different filter types, which run at different stages of request processing:

1. **Authorization Filters**
   * Run first to check if the user is authorized.
   * Example: [Authorize].
2. **Resource Filters**
   * Run after authorization and before model binding.
   * Useful for caching or short-circuiting requests.
3. **Action Filters**
   * Run before and after an action method executes.
   * Useful for input validation, logging, or modifying parameters/results.
4. **Exception Filters**
   * Run when an exception is thrown in the pipeline.
   * Useful for custom error handling.
5. **Result Filters**
   * Run before and after the action result is executed.
   * Useful for modifying response results.

## 🔹 Example 1: Action Filter (Custom Logging)

using Microsoft.AspNetCore.Mvc.Filters;

using System.Diagnostics;

public class LogActionFilter : IActionFilter

{

public void OnActionExecuting(ActionExecutingContext context)

{

Debug.WriteLine(">>> Before executing action: " + context.ActionDescriptor.DisplayName);

}

public void OnActionExecuted(ActionExecutedContext context)

{

Debug.WriteLine(">>> After executing action: " + context.ActionDescriptor.DisplayName);

}

}

👉 Register the filter in Program.cs (global filter):

builder.Services.AddControllersWithViews(options =>

{

options.Filters.Add<LogActionFilter>();

});

👉 Or apply to a controller/action:

[ServiceFilter(typeof(LogActionFilter))]

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

}

## 🔹 Example 2: Exception Filter (Custom Error Handling)

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

context.Result = new JsonResult(new

{

Message = "An error occurred.",

Details = context.Exception.Message

})

{ StatusCode = 500 };

context.ExceptionHandled = true;

}

}

👉 Register globally:

builder.Services.AddControllersWithViews(options =>

{

options.Filters.Add<CustomExceptionFilter>();

});

## 🔹 Example 3: Resource Filter (Caching)

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Mvc;

public class SimpleCacheFilter : IResourceFilter

{

private static readonly Dictionary<string, IActionResult> \_cache = new();

public void OnResourceExecuting(ResourceExecutingContext context)

{

var key = context.HttpContext.Request.Path.ToString();

if (\_cache.ContainsKey(key))

{

context.Result = \_cache[key]; // short-circuit pipeline

}

}

public void OnResourceExecuted(ResourceExecutedContext context)

{

var key = context.HttpContext.Request.Path.ToString();

if (!\_cache.ContainsKey(key))

{

\_cache[key] = context.Result;

}

}

}

## 🔹 Example 4: Result Filter (Modifying Response)

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Mvc;

public class AddHeaderFilter : IResultFilter

{

public void OnResultExecuting(ResultExecutingContext context)

{

context.HttpContext.Response.Headers.Add("X-App-Version", "1.0.0");

}

public void OnResultExecuted(ResultExecutedContext context)

{

// After result execution

}

}

✅ **Summary**

* **AuthorizationFilter** → Check security.
* **ResourceFilter** → Short-circuit requests (e.g., caching).
* **ActionFilter** → Pre/post logic around controller actions.
* **ExceptionFilter** → Centralized error handling.
* **ResultFilter** → Modify response before sending to client.